3.2 Machine-Dependent Loader Features (Linking Loader)

for SIC/XE Machine

Beyond an Absolute Loader

- Shortcoming of an absolute loader
  - Programmer needs to specify the actual address at which it will be loaded into memory.
  - It is difficult to run several programs concurrently, sharing memory between them.
  - It is difficult to use subroutine libraries.
- Solution: a more complex loader that provides
  - Program relocation
  - Program linking

Relocation

- Loaders that allow for program relocation are called *relocating* or *relative loaders*.
- Two methods for specifying relocation as part of the object program
  - Modification records (in chap. 2)
    - Suitable for a small number of relocations required when relative or immediate addressing modes are extensively used
  - Relocation bits
    - Suitable for a large number of relocations required when only direct addressing mode can be used in a machine with fixed instruction format (e.g., the standard SIC machine)

Example of a SIC/XE Program (Fig. 3.4 <= Fig. 2.6)

5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006 07	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013 14		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026 27		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	

5 half bytes

5 half bytes

5 half bytes

Only three addresses need to be relocated.

Example of a SIC/XE Program

```
110              .
115              .          SUBROUTINE TO READ RECORD INTO BUFFER
120              .
125     1036     RDREC      CLEAR     X                  B410
130     1038                CLEAR     A                  B400
132     103A                CLEAR     S                  B440
133     103C               +LDT       #4096              75101000
135     1040     RLOOP      TD        INPUT              E32019
140     1043                JEQ       RLOOP              332FFA
145     1046                RD        INPUT              DB2013
150     1049                COMPR     A,S                A004
155     104B                JEQ       EXIT               332008
160     104E                STCH      BUFFER,X           57C003
165     1051                TIXR      T                  B850
170     1053                JLT       RLOOP              3B2FEA
175     1056     EXIT       STX       LENGTH             134000
180     1059                RSUB                         4F0000
185     105C     INPUT      BYTE      X'F1'              F1
```

Example of a SIC/XE Program

```
195            .
200            .         SUBROUTINE TO WRITE RECORD FROM BUFFER
205            .
210   105D     WRREC     CLEAR    X               B410
212   105F               LDT      LENGTH          774000
215   1062     WLOOP     TD       OUTPUT          E32011
220   1065               JEQ      WLOOP           332FFA
225   1068               LDCH     BUFFER,X        53C003
230   106B               WD       OUTPUT          DF2008
235   106E               TIXR     T               B850
240   1070               JLT      WLOOP           3B2FEF
245   1073               RSUB                     4F0000
250   1076     OUTPUT    BYTE     X'05'           05
255                      END      FIRST
```

Object Program with Modification Records (Fig. 3.5 <= Fig. 2.8)

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05+COPY
M^000014^05+COPY
M^000027^05+COPY
E^000000
```

There is one modification record for each address need to be relocated.

Relocatable Program for SIC
(Fig. 3.6 <= Fig. 2.1)

Fixed instruction format

0000	COPY	START	0	
0000	FIRST	STL	RETADR	140033
0003	CLOOP	JSUB	RDREC	481039
0006		LDA	LENGTH	000036
0009		COMP	ZERO	280030
000C		JEQ	ENDFIL	300015
000F		JSUB	WRREC	481061
0012		J	CLOOP	3C0003
0015	ENDFIL	LDA	EOF	00002A
0018		STA	BUFFER	0C0039
001B		LDA	THREE	00002D
001E		STA	LENGTH	0C0036
0021		JSUB	WRREC	481061
0024		LDL	RETADR	080033
0027		RSUB		4C0000
002A	EOF	BYTE	C'EOF'	454F46
002D	THREE	WORD	3	000003
0030	ZERO	WORD	0	000000
0033	RETADR	RESW	1	
0036	LENGTH	RESW	1	
0039	BUFFER	RESB	4096	

F = 1111

F = 1111

C = 1100

E = 1110

Direct addressing mode

Relocatable Program for SIC

```
                   .
                   .          SUBROUTINE TO READ RECORD INTO BUFFER
                   .
1039     RDREC     LDX        ZERO              040030
103C               LDA        ZERO              000030
103F     RLOOP     TD         INPUT             E0105D
1042               JEQ        RLOOP             30103F
1045               RD         INPUT             D8105D
1048               COMP       ZERO              280030
104B               JEQ        EXIT              301057
104E               STCH       BUFFER,X          548039
1051               TIX        MAXLEN            2C105E
1054               JLT        RLOOP             38103F
1057     EXIT      STX        LENGTH            100036
105A               RSUB                         4C0000
105D     INPUT     BYTE       X'F1'             F1
105E     MAXLEN    WORD       4096              001000
                   .
```

Fixed instruction format

F = 1111

F = 1111

C = 1100

8 = 1000

Direct addressing mode

Relocatable Program for SIC

```
                .       SUBROUTINE TO WRITE RECORD FROM BUFFE
                .
        1061    WRREC   LDX     ZERO            040030
        1064    WLOOP   TD      OUTPUT          E01079
        1067            JEQ     WLOOP           301064
        106A            LDCH    BUFFER,X        508039
        106D            WD      OUTPUT          DC1079
        1070            TIX     LENGTH          2C0036
        1073            JLT     LOOP            381064
        1076            RSUB                    4C0000
        1079    OUTPUT  BYTE    X'05'           05
                        END     FIRST
```

F = 1111

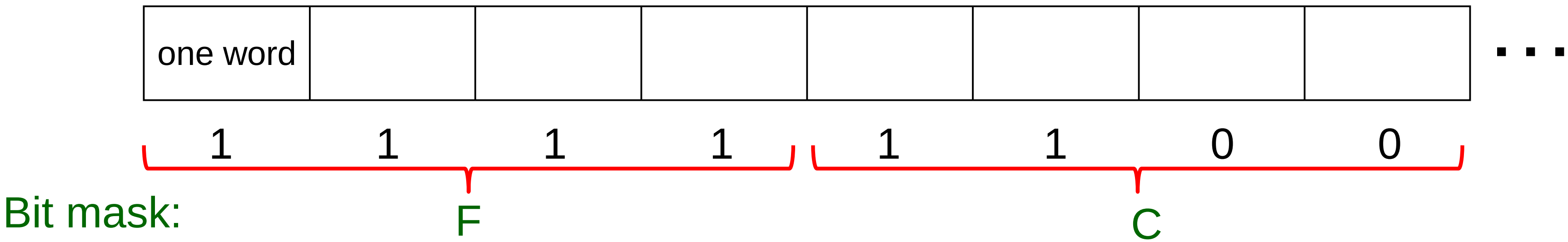
E = 1110

Direct addressing mode

This program does not use relative addressing. Thus the addresses in all the instructions except RSUB must be modified. This would require 31 Modification records.

Relocation Bits

- If there are many addresses needed to be modified, it is more efficient to use a relocation bit, instead of a Modification record, to specify every relocation.
- When the instruction format is fixed as in SIC machine (one word per instruction), we can associate each instruction with a relocation bit.
- Relocation bits can be gathered together into a bit mask to be stored in the Text record.
- If the relocation bit corresponding to a word of object code is set to 1, the program's starting address will be added to this word when the program is relocated.



Object Program with Relocation Bit Mask (Fig. 3.7 <= Fig. 2.3)

```
H^COPY  ^000000^00107A
T^000000^1E^FFC^140033^481039^000036^280030^300015^481061^3C0003^00002A^0C0039^00002D
T^00001E^15^E00^0C0036^481061^080033^4C0000^454F46^000003^000000
T^001039^1E^FFC^040030^000030^E0105D^30103F^D8105D^280030^301057^548039^2C105E^38103F
T^001057^0A^800^100036^4C0000^F1^001000
T^001061^19^FE0^040030^E01079^301064^508039^DC1079^2C0036^381064^4C0000^05
E^000000
```

Why a new record?

Why a new record?

- Relocation bits corresponding to unused words are set to 0.
- The object code 040030 generated from the LDX instruction on line 210 begins a new Text record for proper alignment.

Program Linking

- A program is a logical entity that combines all of the related control sections.
- Control sections could be assembled together, or they could be assembled independently of one another.
- Control sections are to be linked, relocated, and loaded by loaders.
- External references among control sections can be assigned addresses after these control sections are loaded into memory by loaders.

Sample Program for Linking and Relocation (Fig. 3.8)

Loc	Source statement			Object code
0000	PROGA	START	0	
		EXTDEF	LISTA,ENDA	
		EXTREF	LISTB,ENDB,LISTC,ENDC	
		.		
		.		
		.		
0020	REF1	LDA	LISTA	03201D
0023	REF2	+LDT	LISTB+4	77100004
0027	REF3	LDX	#ENDA-LISTA	050014
		.		
		.		
		.		
0040	LISTA	EQU	*	
		.		
		.		
0054	ENDA	EQU	*	
0054	REF4	WORD	ENDA-LISTA+LISTC	000014
0057	REF5	WORD	ENDC-LISTC-10	FFFFF6
005A	REF6	WORD	ENDC-LISTC+LISTA-1	00003F
005D	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000014
0060	REF8	WORD	LISTB-LISTA	FFFFC0
		END	REF1	

Sample Program for Linking and Relocation

```
Loc     Source statement                                   Object code

0000    PROGB    START    0
                 EXTDEF   LISTB,ENDB
                 EXTREF   LISTA,ENDA,LISTC,ENDC
                 .
                 .
                 .
0036    REF1     +LDA     LISTA                            03100000
003A    REF2     LDT      LISTB+4                          772027
003D    REF3     +LDX     #ENDA-LISTA                      05100000
                 .
                 .
                 .
0060    LISTB    EQU      *
                 .
                 .
0070    ENDB     EQU      *
0070    REF4     WORD     ENDA-LISTA+LISTC                 000000
0073    REF5     WORD     ENDC-LISTC-10                    FFFFF6
0076    REF6     WORD     ENDC-LISTC+LISTA-1               FFFFFF
0079    REF7     WORD     ENDA-LISTA-(ENDB-LISTB)          FFFFF0
007C    REF8     WORD     LISTB-LISTA                      000060
                 END
```

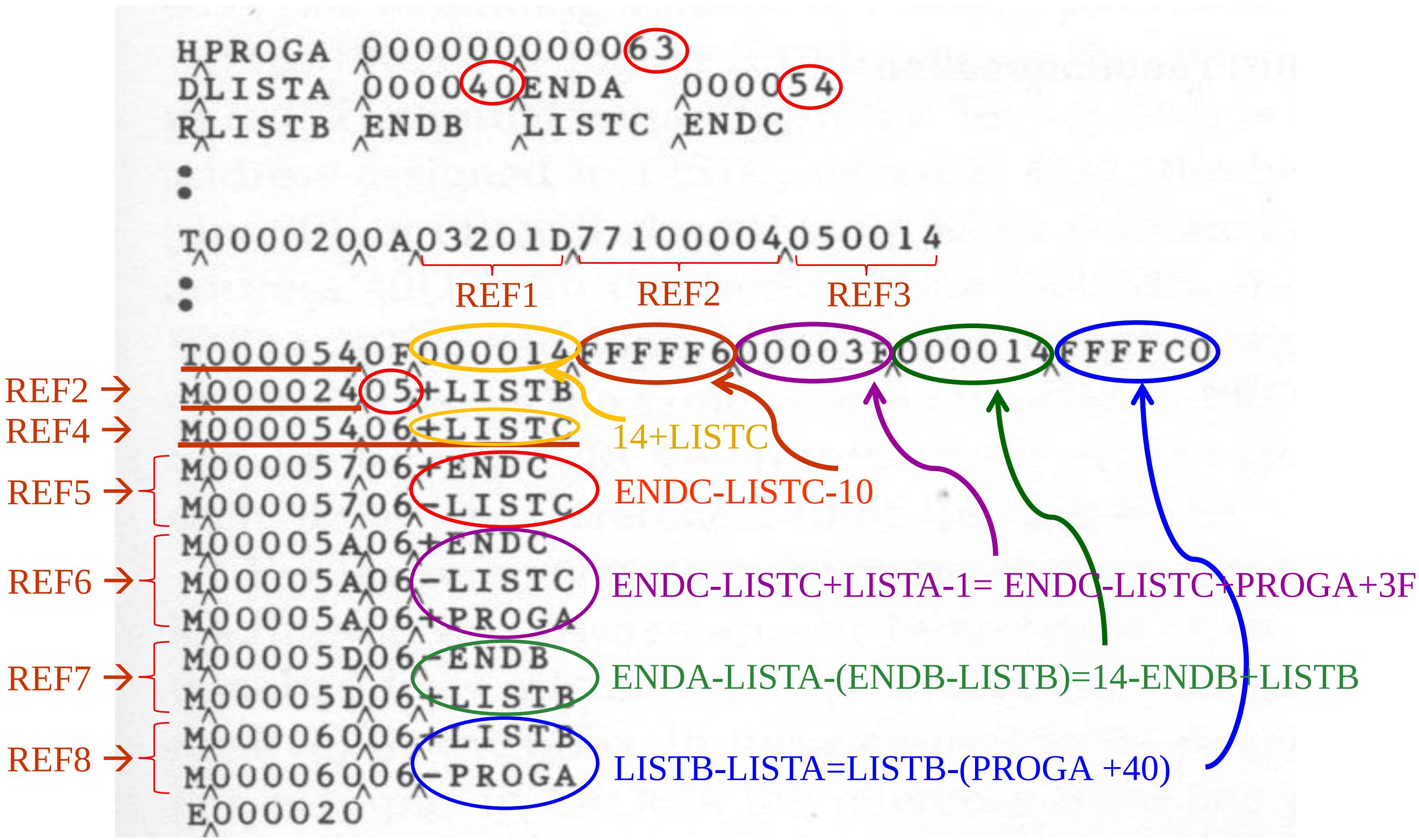
Sample Program for Linking and Relocation

Loc	Source statement			Object code
0000	PROGC	START	0	
		EXTDEF	LISTC,ENDC	
		EXTREF	LISTA,ENDA,LISTB,ENDB	
		.		
		.		
		.		
0018	REF1	+LDA	LISTA	03100000
001C	REF2	+LDT	LISTB+4	77100004
0020	REF3	+LDX	#ENDA-LISTA	05100000
		.		
		.		
		.		
0030	LISTC	EQU	*	
		.		
		.		
0042	ENDC	EQU	*	
0042	REF4	WORD	ENDA-LISTA+LISTC	000030
0045	REF5	WORD	ENDC-LISTC-10	000008
0048	REF6	WORD	ENDC-LISTC+LISTA-1	000011
004B	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000000
004E	REF8	WORD	LISTB-LISTA	000000
		END		

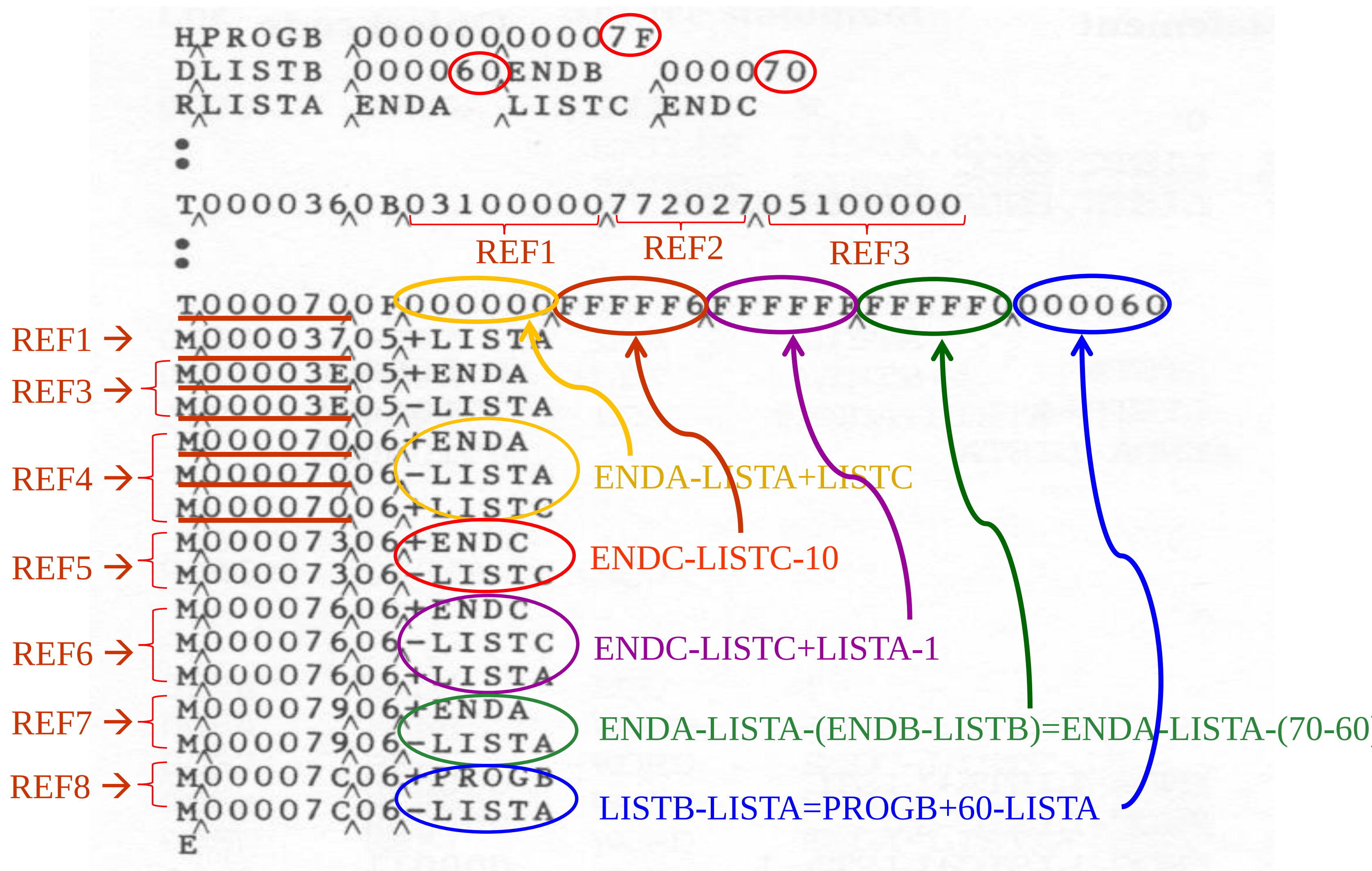
Sample Program for Linking and Relocation

- Each control section defines a list:
  - Control section A: LISTA --- ENDA
  - Control section B: LISTB --- ENDB
  - Control section C: LISTC --- ENDC
- Each control section contains exactly the same set of references to these lists
  - REF1 through REF3: instruction operands
  - REF4 through REF8: values of data words
- After these control sections are linked, relocated, and loaded, each of REF4 through REF8 should have resulted in the same value in each of the three control sections. (but not for REF1 through REF3, why?)

Object Code of Control Section A (Fig. 3.9)



Object Code of Control Section B (Fig. 3.9)



Object Code of Control Section C (Fig. 3.9)

```
H^PROGC ^000000^000051
D^LISTC ^000030^ENDC  ^000042
R^LISTA ^ENDA  ^LISTB ^ENDB
•
•
T^000018^0C^03100000^77100004^05100000
            REF1      REF2      REF3
•
•
T^000042^0F^000030^000008^000011^000000^000000
REF1 → M^000019^05^+LISTA
REF2 → M^00001D^05^+LISTB
REF3 → M^000021^05^+ENDA
       M^000021^05^-LISTA
REF4 → M^000042^06^+ENDA
       M^000042^06^-LISTA
       M^000042^06^+PROGC
REF6 → M^000048^06^+LISTA
REF7 → M^00004B^06^+ENDA
       M^00004B^06^-LISTA
       M^00004B^06^-ENDB
       M^00004B^06^+LISTB
REF8 → M^00004E^06^+LISTB
       M^00004E^06^-LISTA
E
```

ENDC-LISTC-10=12-10=8

ENDA-LISTA+LISTC= ENDA-LISTA+PROGC+30

ENDC-LISTC+LISTA-1=12-1+LISTA

ENDA-LISTA-(ENDB-LISTB)

LISTB-LISTA

External Symbol Table (ESTAB)

PROGA	4000
LISTA	4000+0040=4040
ENDA	4000+0054=4054
PROGB	4000+0063=4063
LISTB	4063+0060=40C3
ENDB	4063+0070=40D3
PROGC	4063+007F=40E2
LISTC	40E2+0030=4112
ENDC	40E2+0042=4124

- Add 40C3 to those five half-byes at 4024 (for REF2 with LISTB).

REF1 (LISTA)

- Control section A
  - LISTA is defined within the control section.
  - Its address is immediately available using PC-relative addressing.
  - No modification for relocation or linking is necessary.
- Control sections B and C
  - LISTA is an external reference.
  - Its address is not available thus an extended-format instruction with address field set to 00000 is used.
  - A modification record is inserted into the object code to instruct the loader to add the value of LISTA (once determined) to this address field(00000).

REF2 (LISTB+4)

- Control sections A and C
  - REF2 is an external reference (LISTB) plus a constant.
  - The address of LISTB is not available thus an extended-format instruction with address field set to 00004 is used.
  - A modification record is inserted into the object code to instruct the loader to add the value of LISTB (once determined) to this address field (00004).
- Control section B
  - LISTB is defined within the control section.
  - Its address is immediately available using PC-relative addressing.
  - No modification for relocation or linking is necessary.

REF3 (#ENDA-LISTA)

- Control section A
  - ENDA and LISTA are defined within the control section.
  - The difference between ENDA and LISTA is immediately available.
  - No modification for relocation or linking is necessary.
- Control sections B and C
  - ENDA and LISTA are external references.
  - The difference between them is not available thus an extended-format instruction with address field set to 00000 is used.
  - Two modification records are inserted into the object code
    - +ENDA
    - -LISTA

REF4 (ENDA-LISTA+LISTC)

- Control section A
  - The values of ENDA and LISTA are known when assembled. Only the value of LISTC is unknown.
  - The address field is initialized as 000014 (ENDA-LISTA).
  - One Modification record is needed for LISTC:
    - +LISTC
- Control section B
  - ENDA, LISTA, and LISTC are all unknown.
  - The address field is initialized as 000000.
  - Three Modification records are needed:
    - +ENDA
    - -LISTA
    - +LISTC
- Control section C
  - LISTC is defined in this control section but ENDA and LISTA are unknown.
  - The address field is initialized as the relative address of LISTC (000030)
  - Three Modification records are needed:
    - +ENDA
    - -LISTA
    - +PROGC (for relocation)

Program in Memory after Linking and Loading (Fig. 3.10(a))

Memory address	Contents				
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	
3FF0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
4000	....	....	....	....	
4010	....	....	....	....	
4020	03201D77	1040C705	0014....	....	←PROGA
4030	....	....	....	....	
4040	....	....	....	....	
4050	....	00412600	00080040	51000004	
4060	000083..	....	....	....	
4070	....	....	....	....	
4080	....	....	....	....	
4090	....	....	..031040	40772027	←PROGB
40A0	05100014	....	....	....	
40B0	....	....	....	....	
40C0	....	....	....	....	
40D0	00	41260000	08004051	00000400	
40E0	0083....	....	....	....	
40F0	....	....	0310	40407710	
4100	40C70510	0014....	....	....	←PROGC
4110	....	....	....	....	
4120	....	00412600	00080040	51000004	
4130	000083xx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
4140	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	

Values of REF4, REF5, …, REF8 in three places are all the same.

PROGA started at 4000

PROGB started at 4063

PROGC started at 40E2

Calculation of REF4 (ENDA-LISTA+LISTC)

- Control section A
  - The address of REF4 is 4054 (4000 + 54)
  - The value of REF4 is:

    000014 + 004112 = 004126

    (initial value) (address of LISTC)
  - The address of LISTC is:

    0040E2 + 000030 = 004112

    (starting address of PROGC) (relative address of LISTC in PROGC)
- Control section B
  - The address of REF4 is 40D3 (4063 + 70)
  - The value of REF4 is:

    000000 + 004054 - 004040 + 004112 = 004126

    (initial value) (address of ENDA) (address of LISTA) (address of LISTC)

Calculation of REF4 (ENDA-LISTA+LISTC)

- Control section C
  - The address of REF4 is 4124 (40E2 + 42)
  - The value of REF4 is:

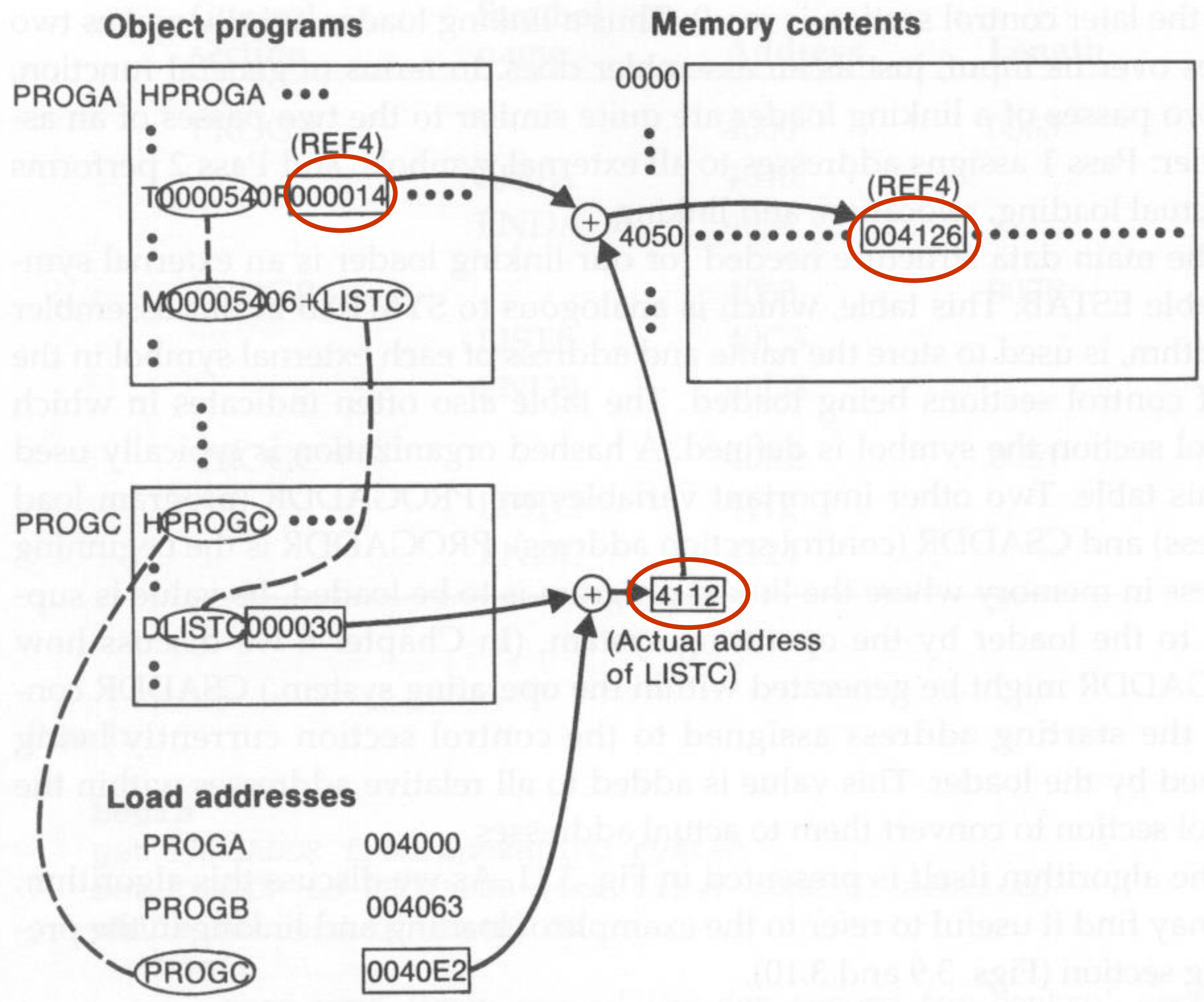
    004054 - 004040 + 004112 = 004126

    (address of ENDA) (address of LISTA) (address of LISTC)
  - The address of LISTC is:

    000030 + 0040E2 = 004112

    (initial value of LISTC in PROGC) (starting address of PROGC)

Calculation of REF4 (ENDA-LISTA+LISTC) (Fig. 3.10(b))

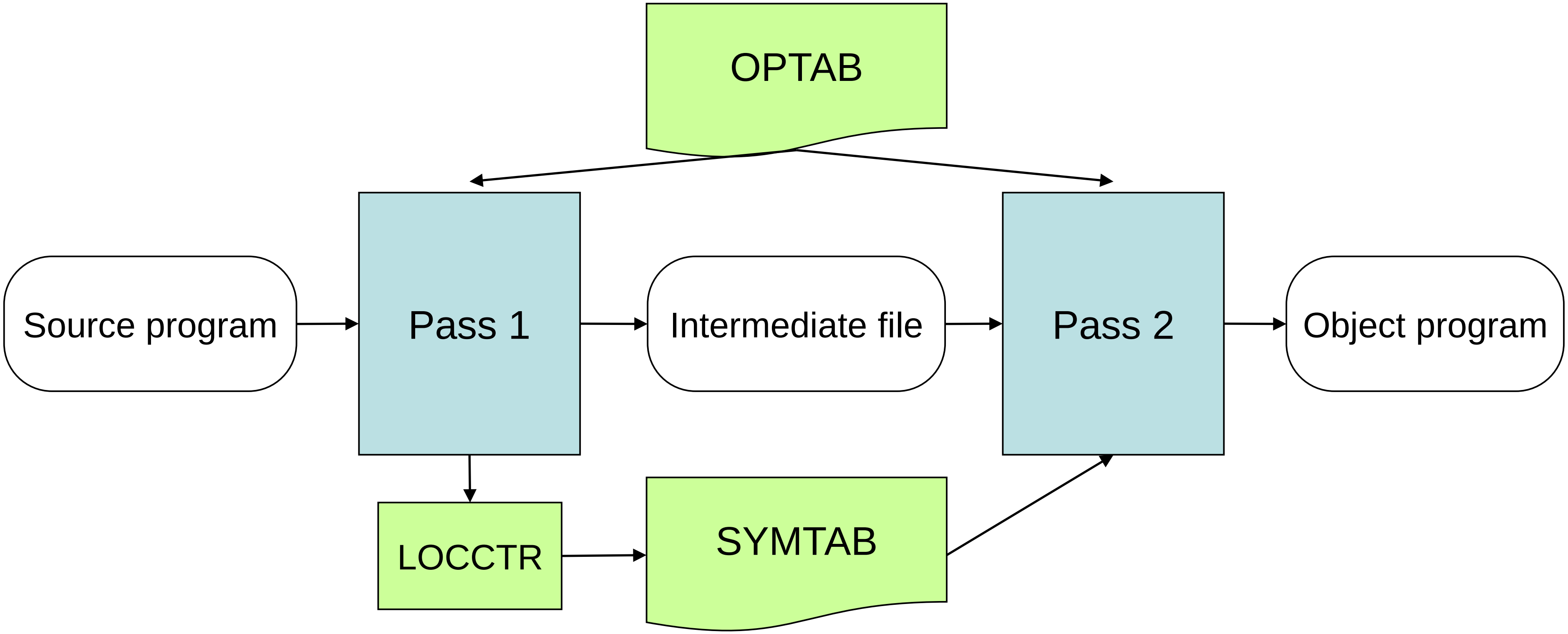


References in Instruction Operands

- For references that are instruction operands, the calculated values after loading do no always appear to be equal.
- This is because there is an additional address calculation step involved for PC (or base) relative instructions.
- In such cases, it is the target addresses that are the same.
- For example, in control section A, the reference REF1 is a PC relative instruction with displacement 01D. When this instruction is executed, the PC contains the value 4023. Therefore the resulting address is 4040. In control section B, because direct addressing is used, 4040 (4000 + 40) is stored in the loaded program for REF1.

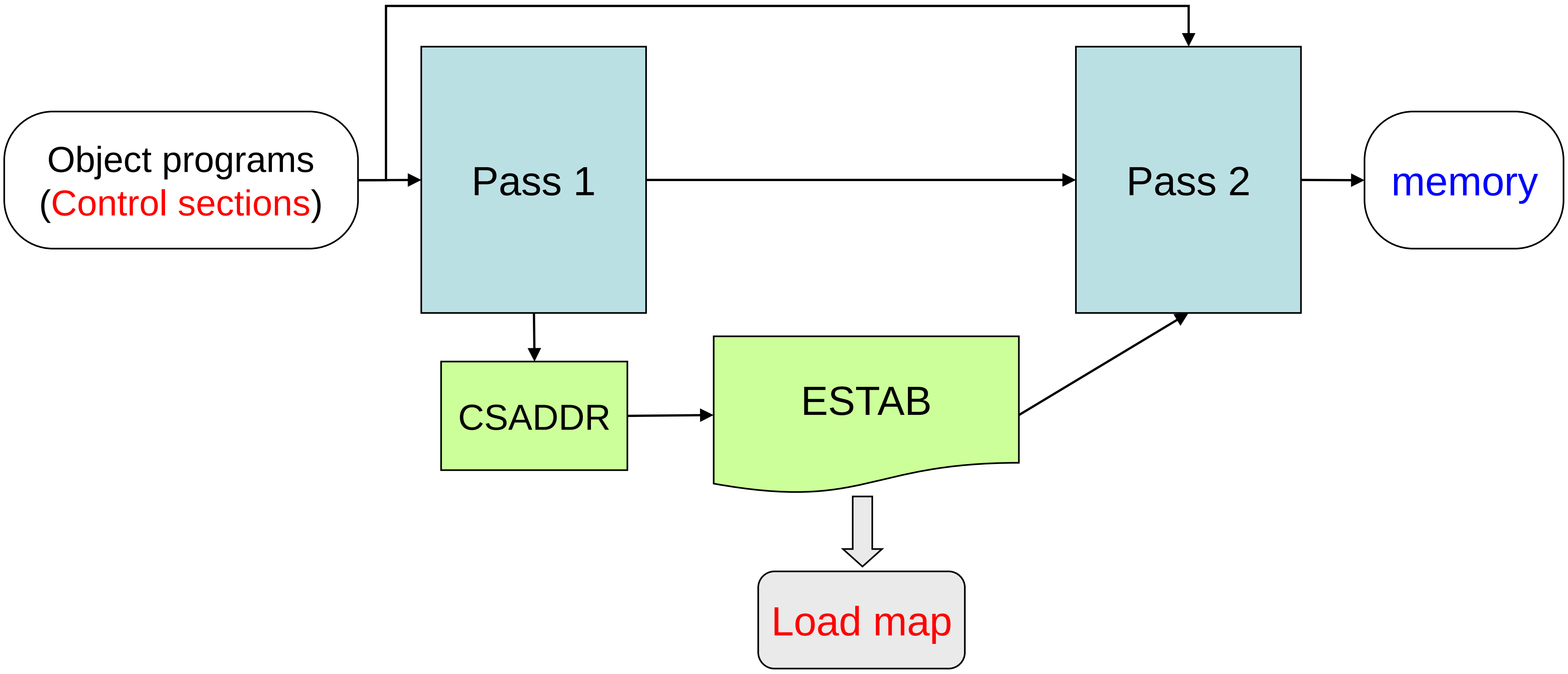
Implementation of an Assembler

- Operation Code Table (OPTAB)
- Symbol Table (SYMTAB)
- Location Counter (LOCCTR)



Implementation of a Linking Loader

- Two-pass process (similar to the Assembler):
  - Pass 1: assigns addresses to all external symbols
  - Pass 2: performs the actual loading, relocation, and linking



Data Structures

- External Symbol Table (ESTAB)
  - For each external symbol, ESTAB stores
    - its name
    - its address
    - in which control section the symbol is defined
  - Hashed organization
- Program Load Address (PROGADDR)
  - PROGADDR is the beginning address in memory where the linked program is to be loaded (supplied by OS).
- Control Section Address (CSADDR)
  - CSADDR is the starting address assigned to the control section currently being scanned by the loader.
  - CSADDR is added to all relative addresses within the control section.

A Load Map

Control section	Symbol name	Address	Length
PROGA		4000	0063
	LISTA	4040	
	ENDA	4054	
PROGB		4063	007F
	LISTB	40C3	
	ENDB	40D3	
PROGC		40E2	0051
	LISTC	4112	
	ENDC	4124	

Algorithm(Fig. 3.11(a))

Pass 1:

(only Header and Define records are concerned)

```
begin
get PROGADDR from operating system
set CSADDR to PROGADDR {for first control section}
while not end of input do
   begin
      read next input record {Header record for control section}
      set CSLTH to control section length
      search ESTAB for control section name
      if found then
         set error flag {duplicate external symbol}
      else
         enter control section name into ESTAB with value CSADDR
      while record type ≠ 'E' do
         begin
            read next input record
            if record type = 'D' then
               for each symbol in the record do
                  begin
                     search ESTAB for symbol name
                     if found then
                        set error flag (duplicate external symbol)
                     else
                        enter symbol into ESTAB with value
                              (CSADDR + indicated address)
                  end {for}
         end {while ≠ 'E'}
      add CSLTH to CSADDR {starting address for next control section}
   end {while not EOF}
end {Pass 1}
```

Algorithm (Fig. 3.11(b))

Pass 2:

```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
   begin
      read next input record  {Header record}
      set CSLTH to control section length
      while record type ≠ 'E' do
         begin
            read next input record
            if record type = 'T' then
               begin
                  {if object code is in character form, convert
                     into internal representation}
                  move object code from record to location
                     (CSADDR + specified address)
               end {if 'T'}
            else if record type = 'M' then
               begin
                  search ESTAB for modifying symbol name
                  if found then
                     add or subtract symbol value at location
                        (CSADDR + specified address)
                  else
                     set error flag (undefined external symbol)
               end  {if 'M'}
         end {while ≠ 'E'}
      if an address is specified {in End record} then
         set EXECADDR to (CSADDR + specified address)
      add CSLTH to CSADDR
   end  {while not EOF}
jump to location given by EXECADDR {to start execution of loaded program}
end {Pass 2}
```

Enhance the Algorithm

- We can make the Assembler more efficient by storing search information in the intermediate file and avoiding the search of OPTAB in Pass 2.
- We can make the linking loader algorithm more efficient by:
  - assigning a reference number to each external symbol referred to in a control section
    - Control section name: 01
    - Other external reference symbols (stored in the Refer records): 02 symname, 03 symname, …
  - using this reference number (instead of the symbol name) in Modification records
  - avoiding multiple searches of ESTAB for the same symbol during the loading of a control section.
    - Search of ESTAB for each external symbol can be performed once and the result is stored in a table **indexed by the reference number**.
    - The values for code modification can then be obtained by simply indexing into the table.

Examples of Using Reference Numbers (Fig. 3.12 <= Fig. 3.9)

```
H^PROGA ^000000^000063
D^LISTA ^000040^ENDA ^000054
R^02LISTB ^03ENDB ^04LISTC ^05ENDC
•
•
T^000020^0A^03201D^77100004^050014
•
•
T^000054^0F^000014^FFFFF6^00003F^000014^FFFFC0
M^000024^05^+02
M^000054^06^+04
M^000057^06^+05
M^000057^06^-04
M^00005A^06^+05
M^00005A^06^-04
M^00005A^06^+01
M^00005D^06^-03
M^00005D^06^+02
M^000060^06^+02
M^000060^06^-01
E^000020
```

Examples of Using Reference Numbers (Fig. 3.12 <= Fig. 3.9)

```
H^PROGB ^000000^00007F
D^LISTB ^000060^ENDB   ^000070
R^02^LISTA ^03ENDA  ^04LISTC ^05ENDC
.
.
T^000036^0B^03100000^772027^05100000
.
.
T^000070^0F^000000^FFFFF6^FFFFFF^FFFFF0^000060
M^000037^05^+02
M^00003E^05^+03
M^00003E^05^-02
M^000070^06^+03
M^000070^06^-02
M^000070^06^+04
M^000073^06^+05
M^000073^06^-04
M^000076^06^+05
M^000076^06^-04
M^000076^06^+02
M^000079^06^+03
M^000079^06^-02
M^00007C^06^+01
M^00007C^06^-02
E
```

Examples of Using Reference Numbers (Fig. 3.12 <= Fig. 3.9)

```
H^PROGC ^000000^000051
D^LISTC ^000030^ENDC  ^000042
R^02^LISTA ^03^ENDA  ^04^LISTB ^05^ENDB
•
•
T^000018^0C^03100000^77100004^05100000
•
•
T^000042^0F^000030^000008^000011^000000^000000
M^000019^05^+02
M^00001D^05^+04
M^000021^05^+03
M^000021^05^-02
M^000042^06^+03
M^000042^06^-02
M^000042^06^+01
M^000048^06^+02
M^00004B^06^+03
M^00004B^06^-02
M^00004B^06^-05
M^00004B^06^+04
M^00004E^06^+04
M^00004E^06^-02
E
```